

# A Survey of Tree Convex Sets Test

Yuanlin Zhang and Forrest Sheng Bao  
Dept. of Computer Science, Texas Tech University  
Lubbock, Texas 79409 \*

## Abstract

Tree convex sets refer to a collection of sets such that each set in the collection is a subtree of a tree whose nodes are the elements of these sets. They extend the concept of row convex sets each of which is an interval over a total ordering of the elements of those sets. They have been applied to identify tractable Constraint Satisfaction Problems and Combinatorial Auction Problems. Recently, polynomial algorithms have been proposed to recognize tree convex sets. In this paper, we review the materials that are the key to a linear recognition algorithm.

## 1 Introduction

Given a set  $U$ , a collection  $S$  of subsets of  $U$  is tree convex if there exists a tree  $T$  with nodes  $U$  such that every set of  $S$  is a subtree (Zhang and Yap, 2003) of  $T$ . Row convex sets are a collection of sets that are tree convex with respect to a chain (a special tree with nodes  $U$ ). Row convex sets correspond to another well studied concept: *consecutive ones property* of matrices. Let  $M$  be the matrix whose rows are indexed by the elements of  $S$  and columns indexed by those of  $U$  in terms of a total ordering over  $U$ . An entry of  $M$ , indexed by  $(s, a)$  with  $s \in S$  and  $a \in U$ , is one if and only if  $a \in s$ .  $M$  has consecutive ones property (Fulkerson and Gross, 1965) with respect to its rows if there is a total ordering of  $U$  such that the ones on each row is consecutive. Clearly, the sets of  $S$  are row convex if and only if the matrix  $M$  has consecutive ones property.

The property of tree convex and row convex sets has been employed to identify tractable Constraint Satisfaction Problems (CSP). CSP problems have found many successful applications in Artificial Intelligence and Combinatorial Problems (Dechter, 2003). However, in general, CSP problems are NP-hard. Continuous research effort has been made to identify tractable CSP problems. An important approach is to make use of semantic properties of the constraints. For *monotone constraints*, path consistency implies global consistency (Montanari, 1974). van Beek and Dechter (1995) generalize monotone constraints to a larger class of *row convex constraints* which is in turn expanded to *tree convex constraints* by Zhang and Yap (2003). The tractability of these constraints results from the nice intersection property of tree convex constraints.

Recently, tree convex sets also have found applications in combinatorial auctions. Given a set  $U$  of items and a collection of bids each of which is a subset of  $U$ , the problem to decide the winners is NP-complete (Rothkopf et al., 1998) in general. However, when the collection of bids are tree convex, the problem becomes tractable (Sandholm and Suri, 2003). (Note that although “tree convexity” is not used in that paper, the concept there is exactly the same as tree convexity.)

An interesting and challenging question raised in the application of tree convex sets in both CSP and Combinatorial Auctions is how efficiently one can test the tree convexity of a given collection of sets. There is abundant related research work under the umbrella of *consecutive ones property test*, i.e., row convexity test. The consecutive ones problem was first proposed by Fulkerson and Gross (1965). A linear algorithm was then developed by Booth and Lueker (1976). It uses quite complex data structures and involved techniques. There exists continuous work, e.g., by Meidanis et al. (1998), Habib et al. (2000), and Hsu (2002), to improve the understanding of consecutive ones property and its test. For tree convexity test, polynomial algorithms have been recently designed by Yosiphon (2003) and Conitzer et al. (2004). Yosiphon makes use of complex data structures and ideas inherited from consecutive ones property work. The resulting algorithm is rather involved and has a complexity of  $O(mn)$ . Conitzer et al. proposes a

---

\* yzhang@cs.ttu.edu, forrest.bao@gmail.com

“simple” algorithm but with a still very high time complexity  $O(mn^2)$  where  $m$  is the number of sets (bids) and  $n$  the number of all distinct elements in the sets, i.e., the number of all items to bid.

A very interesting question is whether there are linear algorithms for tree convexity test like row convexity test. In fact, it is listed as one of the open questions in (Conitzer et al., 2004). This question can be answered positively if we take the collection of sets as a hypergraph. With this perspective, we are not only able to identify a simple and nice characterization of tree convex sets using hypergraphs and properties of hypergraphs, but also to connect this problem with the long line research of conjunctive query evaluation in databases and tree decomposition in Constraint Satisfaction Problems (Beeri et al., 1983; Dechter and Pearl, 1989; Gottlob and Szeider, 2008). As a result, an existing simple and elegant linear algorithm for hypergraphs by Tarjan and Yannakakis (1984) can be directly used to test tree convexity.

Due to a well known example in Constraint Satisfaction Problems where an optimal algorithm AC-4 on enforcing arc consistency does not perform better than a non-optimal algorithm AC-3 (Wallace, 1993) in most cases, we also carry out experiments on a set of randomly generated problems to compare the linear algorithm with the one in (Conitzer et al., 2004). Experimental results show that the former is significantly faster than the latter.

Section 2 reviews basic concepts and terms including those that might have different meanings in different context. The details of a characterization of tree convex sets and related work are given in Section 3. To make this survey self contained, a test algorithm including Tarjan et al.’s algorithm is presented in Section 4. Experimental results are given in Section 5 before we conclude the paper.

## 2 Background

In this section, we will review the basics of tree convex sets, the related concepts of graphs and hypergraphs, and some applications of tree convex sets in Constraint Satisfaction Problems and Combinatorial Auction problems.

A *graph* is a tuple  $(N, E)$  where  $N$  and  $E$  are sets, elements of  $N$  are called vertices or nodes and those of  $E$  edges, and each edge is a set of at most two vertices. Hypergraphs generalize graphs by allowing an edge to be a set of arbitrary number of vertices. Specifically, a *hypergraph*  $H$  is a pair  $(\mathcal{N}, \mathcal{E})$  where  $\mathcal{N}$  is a set of vertices, and  $\mathcal{E}$  consists of nonempty subsets of  $\mathcal{N}$  that are called *hyperedges*. Berge’s book (1973) is an excellent reference for hypergraphs.

### 2.1 Notations and results in graphs

A *clique* of a graph is a set of pairwise adjacent vertices. A graph is *chordal* if every cycle of length at least four has a chord, i.e., an edge joining two nonconsecutive vertices on the cycle. *Forests*, *trees*, *chains* and (*simple*) *path* are defined as usual. To reduce the potential confusion or misunderstanding, we repeat the following definitions. A graph  $(N_1, E_1)$  is a *subgraph* of  $(N, E)$  if  $N_1 \subseteq N$  and  $E_1 \subseteq E$ . Given a tree, a *subtree* is defined as a connected subgraph of the tree. A *forest on a set  $S$*  is a forest whose vertex set is exactly  $S$ .

### 2.2 Notations and results in hypergraphs

We introduce in this section dual hypergraphs, acyclic hypergraphs, join trees and some results on hypergraphs. Throughout this paper, we may use “graphs” for “hypergraphs” and “edges” for “hyperedges” when their meaning is clear from the context.

The *graph*  $G(H)$  of a hypergraph  $H$  is the graph whose vertices are those of  $H$  and whose edges are pairs  $\{x, y\}$  such that  $x$  and  $y$  are in a common edge of  $H$ . A hypergraph  $H$  is *conformal* if every clique of  $G(H)$  is contained in an edge of  $H$ .

The *dual graph*  $H^*$  of a graph  $H = (\{v_1, v_2, \dots, v_n\}, \{S_1, S_2, \dots, S_m\})$  is a hypergraph  $(\{S_1, S_2, \dots, S_m\}, \{R_1, R_2, \dots, R_n\})$  where for  $i \in 1..n$ ,  $R_i = \{S_j \mid v_i \in S_j, j \in 1..m\}$ . The edge  $R_i$  is the set of edges of  $H$  that involve vertex  $v_i$ . Intuitively, one can take  $R_i$  as  $v_i$ .

The acyclicity of a hypergraph involves a sequence of concepts defined below.  $H$  is *reduced* if no edges of it properly contain another edge and every node is in some edge. The *reduction* of  $H$  is  $H$  with any contained edges and non-edge nodes removed.

Let  $H = (\mathcal{N}, \mathcal{E})$  be a hypergraph with nodes  $x$  and  $y$  in  $\mathcal{N}$ . A *path* from  $x$  to  $y$  in  $H$  is a sequence of edges  $E_1, E_2, \dots, E_k$  ( $k \geq 1$ ), such that  $x \in E_1$ ,  $y \in E_k$  and  $E_i \cap E_{i+1} \neq \emptyset$  for  $i \in [1..k-1]$ .  $E_1, E_2, \dots, E_k$  is also called a path from  $E_1$  to  $E_k$ .

Two nodes (or edges) are *connected* if there is a path between them. A set of edges is *connected* if every pair of the edges is connected. A *connected component* of  $H$  is a maximal connected set of edges.

Given a hypergraph and a subset of its nodes, we will now define the “projection” of the graph on these nodes. Let  $M$  be a set of nodes of the hypergraph  $(\mathcal{N}, \mathcal{E})$ . The *set of partial edges generated by  $M$*  is defined to be the reduction of  $\{E \cap M \mid E \in \mathcal{E}\} - \{\emptyset\}$ . It is also called a *node-generated set of partial edges*. Given a set of edges  $\mathcal{F}$ , we say  $(E, F)$ , where  $E, F \in \mathcal{F}$ , is an *articulation pair* if  $E \cap F$  is an *articulation set*, i.e., removing  $E \cap F$  from every edge in  $\mathcal{F}$  strictly increases the number of connected components of  $\mathcal{F}$ .

A *block* of a reduced hypergraph is a connected node-generated set of partial edges without articulation set. A reduced hypergraph is *acyclic* if all its blocks have less than two edges. A hypergraph is said to be *acyclic* if its reduction is.

As examples, consider the graphs in Figure 1(a) and Figure 1(b). The former is acyclic, following our intuition. However, the latter is also acyclic. Although  $a, e, c, a$  form a “cycle,” the graph is acyclic by definition because they the cycle is covered by the edge  $\{a, e, c\}$ .

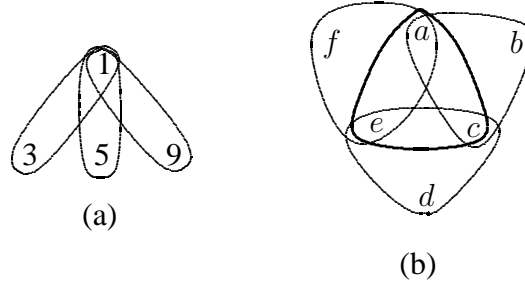


Figure 1: Acyclic graphs can be either tree convex or non tree convex. The letters are the vertices and the edges are represented by enclosed curves.

We define join tree below. Given a collection  $S$  of sets:  $S = \{S_1, S_2, \dots, S_m\}$ , the *intersection graph* for  $S$ , denoted  $I_S$ , is the undirected graph  $(S, E)$  where  $\{S_i, S_j\} \in E$  iff  $S_i \cap S_j \neq \emptyset$ . A path  $S_{i_1}, S_{i_2}, \dots, S_{i_k}$  of  $I_S$  is an *A-path* if  $A \in S_{i_j} \cap S_{i_{j+1}}$  for all  $j \in 1..k-1$ . A subgraph  $G = (S, E')$  of  $I_S$  is a *join graph* if for every pair of nodes  $S_i$  and  $S_j$  of  $S$  and every  $A \in S_i \cap S_j$ , there is an *A-path* from  $S_i$  to  $S_j$  in  $G$ . A *join tree* is a join graph that is a tree. A hypergraph  $(\mathcal{N}, \mathcal{E})$  has a join tree if there is a join tree for  $\mathcal{E}$ . Acyclic graphs and join trees are closely related as revealed by the following result.

**Theorem 1** ((Beeri et al., 1983)). *The following statements on hypergraph  $H$  are equivalent:*

- $H$  is acyclic.
- $H$  has a join tree.
- $H$  is conformal, and  $G(H)$  is chordal.

### 2.3 Tree convex sets

A collection of sets  $S_1, S_2, \dots, S_m$  is *tree convex with respect to a forest  $\mathcal{T}$  on  $\cup_{i=1..m} S_i$*  if every  $S_i$  is a subtree of  $\mathcal{T}$ . For example, the sets  $\{a, b, c\}$ ,  $\{a, b, d\}$ , and  $\{a, c, d\}$  are tree convex with respect to the tree with vertices  $\{a, b, c, d\}$  and edges  $\{\{a, b\}, \{a, c\}, \{a, d\}\}$ .

### 2.4 Tree convex constraints and problems

A binary *constraint network* consists of a set of variables  $V = \{x_1, x_2, \dots, x_n\}$  with a finite domain  $D_i$  for each variable  $x_i \in V$ , and a set of binary constraints  $C$  over the variables of  $V$ .  $c_{xy}$  denotes a constraint on variables  $x$  and  $y$  which is defined as a relation over  $D_x$  and  $D_y$ . Operations on relations, e.g., *intersection* ( $\cap$ ), *composition* ( $\circ$ ), and *inverse*, are applicable to constraints. The *arc* and *path* consistency are defined as in (Mackworth, 1977), and global ( $k$  consistency) consistency in (Freuder, 1978).

Given a constraint  $c_{xy}$ , the *image* of a value  $a$  of  $x$  is the set of values of  $y$  that are compatible with  $a$  under  $c_{xy}$ . A constraint  $c_{xy}$  is *tree convex with respect to a forest  $\mathcal{T}$  on  $D_y$*  if the images of all values of  $D_x$  are tree convex with respect to  $\mathcal{T}$ . A constraint network is *tree convex* if there exists a forest on the domain of each variable such that every constraint  $c_{xy}$  of the network is tree convex with respect to the forest on  $D_y$ .

If a tree convex constraint network is arc and path consistent, it is global consistent (Zhang and Yap, 2003), which implies that a solution can be found in polynomial time.

## 2.5 Combinatorial auction problems

Emerging as key mechanisms for allocating goods, tasks, resources etc., combinatorial auctions (Cramton et al., 2006) allow the bidders to bid on bundles of items, instead of single item. The problem to determine the winners in combinatorial auctions is NP-complete (Rothkopf et al., 1998). However, restricted classes of combinatorial auction problems have been identified. For those classes, there exist efficient polynomial algorithms. We are particularly interested in the class of problems where an item graph of the bids is a tree (Conitzer et al., 2004).

Every *bid* is a set of items. Given a combinatorial auction clearing problem instance (i.e., a set of bids), the graph  $G = (I, E)$ , where  $I$  corresponds to the items in the instance, is a (*valid*) *item graph* if for every bid, the set of items in that bid constitutes a connected subgraph of  $G$ .  $G$  is a *item tree* if it is a tree.

It is straightforward to verify, by the definitions, that a set of bids is tree convex iff there is an item tree for the bids.

Conitzer et al. proposed an algorithm to recognize tree convexity with complexity of  $O(mn^2)$  where  $m$  is the total number of bids and  $n$  the number of total items in the auction. Given a collection of bids  $S = \{S_1, S_2, \dots, S_m\}$ , the algorithm first constructs a graph with vertices  $\cup S (= S_1 \cup S_2 \cup \dots \cup S_m)$ , and weighted edges  $G = \{(\{a, b\}, weight) \mid \exists s \in S \text{ such that } a, b \in s, \text{ and } weight = |\{s \in S : a, b \in s\}|\}$ . It next finds the maximum spanning tree  $T$  of  $G$ .

The sets of  $S$  are tree convex iff the sets are tree convex with respect to  $T$  (Conitzer et al., 2004).

## 3 Characterization of tree convex sets

Given a collection of sets  $S = \{S_1, S_2, \dots, S_m\}$ , let  $U(S) = \cup_{s \in S} s$ . The *hypergraph* of  $S$  is  $(U(S), S)$ . The *dual hypergraph* of  $S$  is the dual graph of  $(U(S), S)$ .

To identify whether  $S$  is tree convex, one convenient way is to look at the hypergraph of  $S$ . Consider the example  $\{\{1, 3\}, \{1, 5\}, \{1, 9\}\}$  in Figure 1(a). Clearly, its hypergraph is acyclic and suggests a tree with respect to which the collection is tree convex. However, we have the following observations about the relationship between a collection of sets and the acyclicity of their hypergraphs.

The graph of  $S$  is acyclic does not necessarily mean the tree convexity of  $S$ . In other words, the graph of a non tree convex sets could be acyclic. Consider the collection  $S = \{\{a, e, f\}, \{c, d, e\}, \{a, b, c\}, \{a, c, e\}\}$  in Figure 1(b). As mentioned before,  $S$  is acyclic. However, it is not tree convex. Assume otherwise it is tree convex with respect to a tree  $T$ . There are paths on  $T$ :  $P_1 : a \rightarrow c$  (because  $a, b$ , and  $c$  form a subtree of  $T$ ),  $P_2 : c \rightarrow e$ ,  $P_3 : e \rightarrow a$ . Clearly,  $P_1 P_2 P_3$  forms a cycle, a contradiction to the fact that  $T$  has no cycles. Another observation is that not all tree convex sets form an acyclic hypergraph. The example  $S = \{\{a, b, c\}, \{a, b, d, e\}, \{b, c, d\}\}$  (Figure 2(a)), given by Yosiphone<sup>1</sup>, is tree convex but not acyclic. Each set of  $S$  is a subtree of the tree shown in the figure. From the intersection graph of  $S$  in Figure 2(b), there does not exist a join tree for  $S$ . So,  $S$  is not acyclic.

In fact, the tree convexity of a collection is related to the acyclicity of its dual graph.

**Theorem 2.** *A collection  $S$  of sets is tree convex iff its dual hypergraph is acyclic.*

*Proof.* Given a collection  $S$  of sets, let  $H = (\{v_1, v_2, \dots, v_n\}, S)$  be its hypergraph. Here we take  $U(S)$  as  $\{v_1, v_2, \dots, v_n\}$ . Let  $D = (S, \{R_1, R_2, \dots, R_n\})$  be the dual graph of  $S$ .

*Necessary condition.* Let  $T$  be a tree on  $U(S)$  such that  $S$  is tree convex with respect to it. The idea is to construct a join tree for  $D$  so that  $D$  is acyclic by Theorem 1. We now construct a tree  $T' = (V, E)$  where  $V = \{R_1, R_2, \dots, R_n\}$ . For all  $R_i, R_j \in V$ ,  $\{R_i, R_j\} \in E$  if and only if  $\{v_i, v_j\}$  is an edge of  $T$ . We next show that  $T'$  is a join tree for  $D$ . Consider any two vertices  $R_i$  and  $R_j$  such that  $R_i \cap R_j \neq \emptyset$  and any  $f \in R_i \cap R_j$  (note  $f$  is an edge of  $H$ ). By definition of dual graph,  $v_i, v_j \in f$  because  $f \in R_i \cap R_j$  and  $R_i$  and  $R_j$  consist of edges involving  $v_i$  and  $v_j$  respectively. There is a unique path from  $v_i$  to  $v_j$  in  $T$ . Let it be  $P = v_i, v_{i+1}, \dots, v_j$ .  $S$  is tree convex implies  $f$  is a subtree of  $T$ . Since both  $v_i$  and  $v_j$  belong to  $f$ , all vertices on  $P$  are in  $f$ . Corresponding to  $P$ , there is a path

<sup>1</sup>Personal communication 2004.

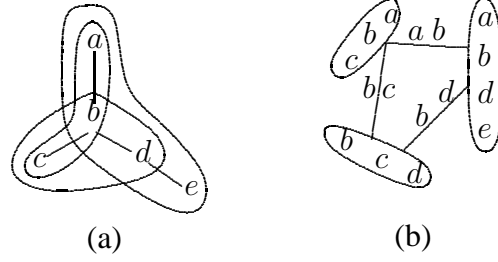


Figure 2: Tree convex sets might not be acyclic. (a) Straight lines represent edges of the underlying tree on the vertices. (b) Enclosed curves represent nodes which correspond to edges in (a). Letters on the straight edges represent the intersection of the nodes at their ends.

$P' = R_i, R_{i+1}, \dots, R_j$  in  $T'$  by the construction of  $T'$ . For all  $k \in i..j$ , since  $v_k \in f$ , we have  $f \in R_k$ . Hence,  $P'$  is an  $f$ -path from  $R_i$  to  $R_j$ . Therefore,  $T'$  is a join tree of  $D$ .

Sufficient condition. Since the dual graph of  $S$  is acyclic, there is a join tree  $T' = (\{R_1, R_2, \dots, R_n\}, R)$  for  $D$  by Theorem 1. We will show that there is a tree  $T$  under which  $S$  is tree convex. Construct  $T = (\{v_1, v_2, \dots, v_n\}, E)$  where  $(v_i, v_j) \in E$  if and only if  $\{R_i, R_j\} \in R$ . Clearly,  $T$  is a tree. We next prove that for any  $s \in S$ ,  $s$  is a subtree of  $T$ . Specifically, we show that for any two vertices  $v_i$  and  $v_j$  of the edge  $s$ , there exists a path from  $v_i$  to  $v_j$  in  $T$  and the nodes on the path are in  $s$ . By definition of dual graphs,  $s \in R_i$  and  $s \in R_j$  because  $v_i, v_j \in s$ . Since  $T'$  is a join tree of  $D$ , there is an  $s$ -path from  $R_i$  to  $R_j$ :  $R_i, R_{i+1}, \dots, R_j$  in  $T'$ . By the construction of  $T$ ,  $v_i, v_{i+1}, \dots, v_j$  is a path of  $T$ . For all  $k \in 1..j$ , since  $s \in R_k$ , we have  $v_k \in s$ . Hence,  $s$  is a subtree of  $T$  and thus  $S$  is tree convex.  $\square$

To illustrate the concepts used in the proof, consider the collection  $S = \{\{a, b, c\}, \{a, b, d, e\}, \{b, c, d\}\}$  again. Let  $e_1 = \{a, b, c\}$ ,  $e_2 = \{a, b, d, e\}$ , and  $e_3 = \{b, c, d\}$ . The hypergraph of  $S$  is  $H = (\{a, b, c, d, e\}, \{e_1, e_2, e_3\})$  (Figure 2(a)). The dual graph of  $S$  is  $D = (\{e_1, e_2, e_3\}, \{R_a, R_b, R_c, R_d, R_e\})$  (Figure 3(a)) where  $R_a = \{e_1, e_2\}$ ,  $R_b = \{e_1, e_2, e_3\}$ ,  $R_c = \{e_1, e_3\}$ ,  $R_d = \{e_2, e_3\}$ ,  $R_e = \{e_2\}$ . Since  $R_e$  is a subset of  $R_d$  and other edges are subsets of  $R_b$ , we have a join tree shown in Figure 3(b). So,  $D$  is acyclic. From the join tree, we can construct a tree on the nodes of the original sets as in Figure 3(c).  $S$  is tree convex with respect to the tree.

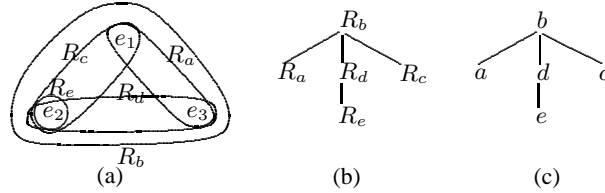


Figure 3: (a) The dual graph of  $S$ . Every edge has a label of  $R$  with subscript. (b) A join tree. (c) Tree derived from (b). The nodes are the elements in the original sets.

A result similar to Theorem 2 was discovered by Goodman and Shmueli (1983) long time ago in the study of database schemas. They provided a rather comprehensive characterization of acyclic hypergraphs. One of their main results is the relationship between acyclic hypergraph and chordality and conformality which is well known by the constraint community (Beeri et al., 1983; Dechter, 2003). However, another result is not known well but directly related to the characterization of tree convexity. It is worth reviewing the result here. First, we introduce some of their terms that are not well known in the constraint community. In the case that confusion could arise from the use of common terminologies, we underline the terms.

Given a hypergraph  $H = (\mathcal{N}, \mathcal{E})$ , a dual graph for  $H$  (Goodman and Shmueli, 1983) is a graph  $G = (V_{\mathcal{E}}, F)$  equipped with a one one onto map  $V_{\mathcal{E}}$  to  $\mathcal{E}$  indicating which node of  $G$  represents which edge of  $\mathcal{E}$ . Note that  $G$  is not a hypergraph here, but just a graph. One type of dual graph used by Goodman and Shmueli is an intersection graph, denoted by  $\Omega(H)$ .  $\Omega(H) = (V_{\mathcal{E}}, F)$  such that  $\{x, y\} \in F$  iff  $E_x \cap E_y \neq \emptyset$  where  $E_x$  and  $E_y$  are the edges (of  $H$ ) represented by  $x$  and  $y$  respectively. A second type of dual graph is a qual graph (Bernstein and Goodman, 1981). Given  $u \in \mathcal{N}$ , the dual of  $u$  is  $u^* = \{E \in \mathcal{E} \mid u \in E\}$ . A qual graph for  $H$  is any dual graph  $G = (V_{\mathcal{E}}, F)$  such that

for each  $u \in \mathcal{N}$ , the subgraph of  $G$  induced by nodes representing elements of  $u^*$  is connected. One can verify that the graph of Figure 3(c) is a qual graph of the hypergraph of Figure 3 (a). The nodes  $a$  to  $e$  of Figure 3(c) represent edges  $R_a$  to  $R_e$ . As an example, consider node  $e_3$ . Its dual  $e_3^* = \{R_c, R_b, R_d\}$ . The subgraph of Figure 3(c) induced by  $a, b, c$  (representing the elements of  $e_3^*$ ) is connected.

A database *schema* can be thought of as a hypergraph whose nodes are the schema's attributes and whose edges are the schema's relations. A hypergraph  $H$  is a *tree schema* if some qual graph for it is a tree.

Now we are ready to present Goodman and Shmueli's result (Goodman and Shmueli, 1983, Theorem 6).

**Theorem 3** (Goodman and Shmueli 1983). *A hypergraph  $H$  is a tree schema iff  $H$  is acyclic.*

Theorem 2 and 3 are equivalent. First, One can show that if a collection of sets is tree convex with respect to a forest, it is tree convex with respect to a tree, and vice versa. Next, by Theorem 2, hypergraph  $H$  is acyclic iff the collection of the edges of its dual graph,  $H^*$ , is tree convex. Thirdly, a key observation is that the collection of edges of  $H^*$  is tree convex iff some qual graph for  $H$  is a tree. By the definition of tree convexity, the former condition holds iff there exists a tree  $T$  with nodes of  $H^*$  such that every edge of  $H^*$  is a subtree of  $T$ . Clearly, by the definition of qual graph,  $T$  is a qual graph for  $H$ . Finally, by definition of tree schema,  $H$  is a tree schema iff there exists a qual graph for  $H$ .

Recently, a nice and more general result on hypergraphs was discovered by Gottlob and Greco (Gottlob and Greco, 2007).

**Theorem 4** (Gottlob and Greco 2007). *Let  $k$  be a number and  $H = (\mathcal{N}, \mathcal{E})$  a hypergraph such that for each node  $v \in \mathcal{N}$ ,  $\{v\} \in \mathcal{E}$ . Then, a  $k$ -width tree decomposition of an item graph for  $H$  exists if and only if  $H^*$  has a  $(k+1)$ -width strict hypertree decomposition.*

Essentially, the hypergraph  $H$  is a set of bids (i.e., a collection of sets). A detailed explanation of the concepts of  $k$ -width tree decomposition of a graph and  $(k+1)$ -width (strict) hypertree decomposition of a hypergraph can be found in (Gottlob and Greco, 2007). This result relates a more general property of a hypergraph with some property of the its dual. A 1-width tree decomposition of an item graph for  $H$  exists if and only if an item graph for  $H$  is a tree, i.e.,  $H$  is tree convex. By definition of strict hypertree decomposition, one can show that a hypergraph has a 2-width strict hypertree decomposition if and only if it is acyclic. So, Theorem 4 implies Theorem 2 and thus 3.

**Remark.** Given a hypergraph  $H$  (representing the topological structure of a CSP problem), its dual (constraint) graph is defined as the intersection graph for  $H$  in (Dechter, 2003). Clearly, the dual graph is different from dual graph and dual (constraint) graph. The definition of intersection graph agrees with that of intersection graph. As for the definitions of acyclic graphs, we follow those in (Beeri et al., 1983). Acyclic hypergraphs are called hypertrees in (Dechter, 2003), but  $\alpha$ -acyclic graphs in (Fagin, 1983) where other types of acyclicity are also introduced.

## 4 Algorithms to identify tree convexity

By Theorem 2, we have the following algorithm to test the tree convexity of a given collection  $S$  and produce a tree if the given collection is tree convex.

---

**Algorithm 1:** Recognize tree convexity of sets

---

```

isTreeConvex(in  $S$ )
1 Let  $D$  be the dual graph of  $S$ 
2 if isAcyclic( $D, R, \gamma$ ) then
3   | genForest( $D, R, \gamma, T$ )
4   | return (true,  $T$ )
   else
5   | return false

```

---

The algorithm first constructs the dual graph  $D$  of  $S$ . The function `isAcyclic( $D, R, \gamma$ )` returns true and data structures  $R$  and  $\gamma$  (discussed below) if the graph of  $D$  is acyclic, and it returns false otherwise. In the former case, using  $R$  and  $\gamma$ , `genForest( $D, R, \gamma, T$ )` builds tree  $T$  (using  $R$  and  $\gamma$ ) with respect to which  $S$  is tree convex.

Based on the work by Rose et al. (1976), Tarjan and Yannakakis (1984) proposed a simple linear algorithm (maximum cardinality search) to identify whether a hypergraph is acyclic. Although maximum cardinality search on a graph can be easily found in a wide range of references (Dechter, 2003), very few references involve the search over hypergraphs. We include it here to make our presentation complete, with the correction of some errors in the original presentation.

Given a graph  $(\mathcal{N}, \mathcal{E})$ , the key behind this algorithm is to compute three mappings  $\alpha$ ,  $\beta$ , and  $\gamma$ . A mapping is a (possibly partial) function that assigns a node and/or an edge to a number between (including) 1 and  $|\mathcal{N}|$ . Specifically, the domain of  $\alpha$  is  $\mathcal{N}$ , that of  $\beta$  is  $\mathcal{N}$  and  $\mathcal{E}$ , and that of  $\gamma$  is  $\mathcal{E}$ . The algorithm, called *restricted maximum cardinality search on hypergraph*, works as follows. It first selects an edge  $s$  from  $\mathcal{E}$  arbitrarily. Mapping  $\alpha$  assigns the nodes of  $s$  the number from  $n$  to  $n - |s| + 1$  one by one. An edge is *exhausted* if all of its nodes have been assigned a number by  $\alpha$ , and *nonexhausted* otherwise. Next we select a nonexhausted edge  $t$  with the maximum number of nodes assigned by  $\alpha$  (tie will be broken arbitrarily). Let  $n_1$  be the largest number that is smaller than  $|\mathcal{N}|$  but not used by  $\alpha$  yet. Assign the non-assigned nodes of  $t$  to numbers from  $n_1$  to  $n_1 - |t| + 1$ . Repeat this process until every node of the graph is assigned a number by  $\alpha$ .  $R(i)$  is used to remember the  $i^{th}$  selected edge. The mapping  $\beta$  is defined as follows. If  $s$  is the  $i^{th}$  selected edge,  $\beta(s) = i$ . Otherwise, it is not defined. For a node  $v$ ,  $\beta(v)$  is defined as  $\beta(s)$  where  $s$  is the first selected edge such that  $v \in s$ , i.e.,  $\beta(v) = \min\{\beta(s) \mid s \text{ is selected and } v \in s\}$ . (Note that in line 12 of the algorithm,  $\beta(E) \leftarrow k$  is redundant. We keep it there to make it compatible with the original algorithm. It also makes the definition of  $\beta$  clearer.) For each edge  $s$ , if  $s$  is not selected during the process,  $\gamma(s)$  is  $\beta(v)$  where  $v \in s$  is the last one to be assigned a number by  $\alpha$ , i.e.,  $\gamma(s) = \max\{\beta(v) \mid v \in s\}$ ; if  $s$  is selected by the process,  $\gamma(s)$  is  $\beta(v)$  if  $v \in s$  is the last node assigned by  $\alpha$  strictly before  $s$  is selected, i.e.,  $\gamma(s) = \max\{\beta(v) \mid v \in s \text{ and } \beta(v) < \beta(s)\}$ , in the last case, if  $\beta(v) = \beta(s)$  for all  $v \in s$ ,  $\gamma(s)$  is not defined.

The mappings are then employed to test the acyclicity of a graph. Given a hypergraph  $H$ , assume totally  $k$  edges are selected during the process above.  $H$  is acyclic iff for each  $i \in 1..k$  and each edge  $s$  such that  $\gamma(s) = i$ ,  $s \cap \{v \mid \beta(v) < i\} \subseteq R(i)$ . The code from line 26 to 32 implements this test.

To compute the mappings in linear time, data structures  $set(i)$ ,  $size(s)$  and  $j$  are maintained during the process of building  $\alpha$ . For each  $s$ ,  $size(s)$  is the count of assigned vertices in  $s$  if  $s$  is nonexhausted and  $-1$  otherwise. For  $i \in 0..n-1$ ,  $set(i)$  is the set of nonexhausted edges that have exactly  $i$  assigned vertices by  $\alpha$ . Index  $j$  is the maximum  $i$  such that  $set(i)$  is nonempty.

The algorithms to test acyclicity and generate the forest are of linear time complexity (Tarjan and Yannakakis, 1984). Hence, we have the following result.

**Theorem 5.** *The worst case time complexity of the algorithm to identify the tree convexity of a collection of sets is linear in the problem size.*

Given a collection of sets  $S = \{S_1, S_2, \dots, S_m\}$ , the size of the problem is  $\sum_{i=1}^m (|S_i|)$ . The complexity of the acyclicity based algorithm is linear to the problem size. Conitzer et al.'s algorithm has a complexity of  $O(mn^2)$  where  $n = |\cup S|$ . Note that the size of each set (bid) may range from 1 to  $n$ , but never exceeds  $n$ . So, the difference of the worst case complexity of the two algorithms is clear.

Algorithm 2 differs from that of (Tarjan and Yannakakis, 1984) in the following two parts. 1) Line 14 was  $i++$  in the original paper, which was clearly a typo. 2) Instead of having line 22-23, the original algorithm increases  $j$  by one right before line 25, which is not correct. Our newly added code in line 22-23 will preserve the linear complexity of the algorithm. In the complexity analysis, line 25 is the key. The number of executions of line 25 during the whole process can be taken as a combination of two parts: executions caused by the monotonic decrease of  $j$ , and those extra executions  $d$  caused by the increase of  $j$  in line 22-23.  $d$  is  $n$  in the worst case as every node of  $U(\mathcal{E})$  will be selected once and only once and for each selected node  $d$  will be increased by only one in the worst case. The new change follows the amortization spirit used in the original analysis. Therefore, Algorithm 2 still has linear complexity.

In the following comment, we use the notations and refer to the original algorithm (page 573) in (Tarjan and Yannakakis, 1984). In a personal communication, Yanakakis and Tarjan points out two alternatives to correct the original algorithm. The first is to replace  $j := j + 1$  by  $j := |R(k)|$ . The other way is to move  $j := j + 1$  to the line immediately before the inner for loop, i.e., line 15, where  $i$  is updated.

---

**Algorithm 2:** Acyclicity test and generation of the forest

---

```
isAcyclic(in  $\mathcal{E}$ , out  $R$ ,  $\gamma$ )
1 Let  $n$  be the number of nodes in  $U(\mathcal{E})$ 
2 for each  $i \in 0..n-1$  do
3    $set(i) \leftarrow \emptyset$ 
4 for  $E \in \mathcal{E}$  do
5    $size(E) \leftarrow 0$ 
6    $\gamma(E) \leftarrow undefined$ 
7   add  $E$  to  $set(0)$ 
8  $i \leftarrow n+1, j \leftarrow 0, k \leftarrow 0$ 
9 while  $j \geq 0$  do
10  delete any  $E$  from  $set(j)$ 
11   $k++$ 
12   $\beta(E) \leftarrow k, R(k) \leftarrow E, size(E) \leftarrow -1$ 
13  for  $v \in E$  such that  $\alpha(v)$  is not assigned do
14     $i--$ 
15     $\alpha(v) \leftarrow i, \beta(v) \leftarrow k$ 
16    for  $F \in \mathcal{E}$  such that  $v \in F$  and  $size(F) \geq 0$  do
17       $\gamma(F) \leftarrow k$ 
18      delete  $F$  from  $set(size(F))$ 
19       $size(F)++$ 
20      if  $size(F) < |F|$  then
21        add  $F$  to  $set(size(F))$ 
22        if  $j < size(F)$  then
23           $j \leftarrow size(F)$ 
24      else
25         $size(F) \leftarrow -1$ 
26  while  $j \geq 0$  and  $set(j) = \emptyset$  do  $j--$ 
27 for  $v \in U(\mathcal{E})$  do  $index(v) \leftarrow 0$ 
28 for each  $i \in 1..k$  do
29   for  $v \in R(i)$  do  $index(v) \leftarrow i$ 
30   for each  $E \in \mathcal{E}$  such that  $\gamma(E) = i$  do
31     for  $v \in E$  do
32       if  $\beta(v) < i$  and  $index(v) < i$  then
33         return false
34 return true
35 genForest
36 genForest(in  $\mathcal{E}$ ,  $R$ ,  $\gamma$ , out  $\mathcal{T}$ )
37  $V \leftarrow \mathcal{E}$ 
38  $E \leftarrow \{ \{F, R(\gamma(F))\} \mid$ 
39    $F \in \mathcal{E} \text{ and } \gamma(F) \text{ is defined} \}$ 
40  $\mathcal{T} \leftarrow (V, E)$ 
```

---

## 5 Experimental evaluation

We have carried out an experimental evaluation of the performance of the acyclicity based algorithm and the spanning tree based algorithm (Conitzer et al., 2004). The algorithm in (Conitzer et al., 2004) consists of two parts: the first part is to find a tree over the items (see the background section) and the second part is to test whether every set (bid) is a subtree of the constructed tree. Due to space limitation, no concrete algorithm for the second part is provided in



(Conitzer et al., 2004). However, it is mentioned in (Conitzer et al., 2004) that the missed algorithm is achievable in  $O(mn)$  where  $m$  is the number of sets (bids), and  $n$  the number of elements (items). To make this paper complete and the experiments here reproducible, we include an algorithm for the second part. The idea is to get the subgraph of the tree induced from each set (line 1-4) and then check the connectedness of each induced graph (line 5-6).

---

**Algorithm 3:** Identify tree convex sets with respect to a given tree

---

```

treeTest(in  $S, T$ )
1 for each  $s \in S$  do construct graph  $G_s = (s, \emptyset)$ 
2 for each edge  $\{a, b\}$  of  $T$  do
3   for each  $s \in S$  do
4     if  $\{a, b\} \in s$  then
        $\hookrightarrow$  add edge  $\{a, b\}$  to graph  $G_s$ 
5 for each graph  $G_s$  do
6   if the connected component of  $G_s$  is not equal to  $s$  then
        $\hookrightarrow$  return false
7 return true

```

---

For line 6, the connected component of a graph can be identified in linear time (Cormen et al., 1990). The complexity of the algorithm is  $O(mn)$  due to the two loops (line 2 and 3).

Recall that a collection of sets, i.e., a set of bids, is *tree convex* iff there is an *item tree* for the bids. So the algorithm in (Conitzer et al., 2004) is directly applicable to tree convexity test and thus no modification or reconstruction is necessary. Our implementation is faithful to the algorithm given in (Conitzer et al., 2004). The experiments are carried out on an AMD Opteron 2350 CPU (frequency 2.0 GHz) with Ubuntu Linux 9.04 of kernel 2.6.28-11. The algorithms are implemented using Python 2.6.2.

From our implementation, we have the following comments about the simplicity of the algorithms. Both algorithms are conceptually quite simple. However, as for implementation, we find that the pseudo code and data structures of the acyclicity algorithm can be “directly” implemented. When we implement the spanning tree based algorithms we have to choose the data structures on graphs carefully so that all the complexity results follow. The final implementation code is much more complex and longer than that of the acyclicity based algorithm.

Acyclic based and spanning tree based algorithms are evaluated on random problems (generated by ourselves) and the structured problems provided by Leyton-Brown et al. (2000).

## 5.1 Random problems

Four parameters are employed to generate our own collections of sets:  $\langle m, n, r_1, r_2 \rangle$  where  $m$  denotes the number of sets of the collection to generate, the size of the sets is between  $r_1$  and  $r_2$ , and each set takes values from 1 to  $n$ .

The evaluation is designed as follows. Since the acyclicity based algorithm is theoretically faster than the spanning tree based algorithm, for large problems, its practical performance should also be faster. We sample a few problems with large configuration parameters to show how the difference between these two algorithms could be. From Table 1 where the time is for 10 problem instances, the acyclicity based algorithm is one to two orders of magnitude faster than that of the spanning tree based algorithm. As the problem size grows, the cost of spanning tree based algorithm grows much faster than that of the acyclicity based algorithm.

$m$	$n$	$r_1$	$r_2$	Acyclicity based	Spanning tree based
100	100	2	10	0.05	1.03
300	300	2	30	0.21	15.99
500	500	2	50	0.56	69.40

Table 1: Performance for large parameters

For small problems, theoretical time complexity might not fully agree with practical performance. Therefore, we employ a systematic comparison scheme: vary the value of  $m$  and  $r_2$  respectively with other parameters fixed.

Specifically, we have tested the following configurations  $\langle m, 100, 2, r_2 \rangle$  where  $m$  changes from 10 to 200 with a step of 10, and  $r_2$  changes from 20 to 90 with step 10. 100 instances are generated from each configuration of the parameters. Samples of the results are shown in Figure 4 and Figure 5.

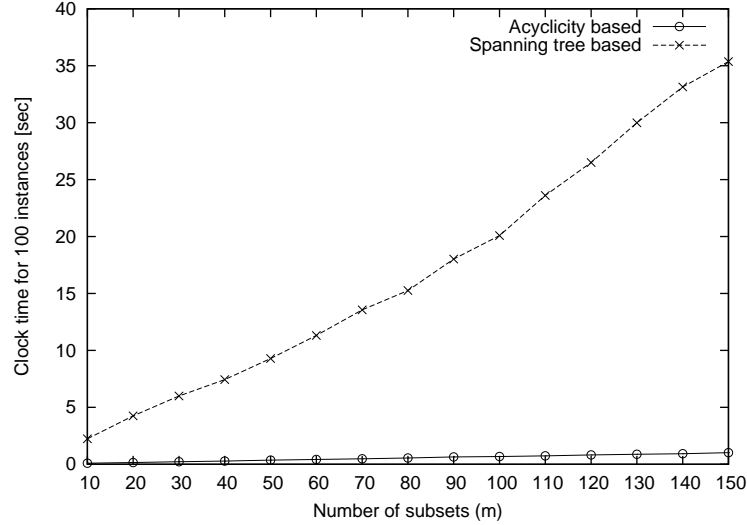


Figure 4: Performance of the algorithms on problems  $\langle m, 100, 2, 30 \rangle$  with  $m$  changing from 10 to 200 with a step of 10

From the results, the acyclicity algorithm runs significantly faster than the spanning tree based algorithm.

## 5.2 Existing structured problems

The problems (Leyton-Brown et al., 2000) used in our experiments are *arbitrary*, *matching*, *paths*, *regions*, *scheduling* and *Legacy* (L1-L8). Their instances are generated from the program at <http://www.cs.ubc.ca/~kevinlb/CATS/>. The details of the description of these problems can be found at (Leyton-Brown et al., 2000). Each problem instance is a set of bids. Our task is to check the tree convexity of the bids. The results are listed in Table 2. In the table, each time entry is for 50 instances. From Table 2, the acyclicity based algorithm is 30 to 80 times faster than the spanning tree base algorithm. It is worth of mentioning that all the instances in the benchmarks are not tree convex, which partially justify our use of random problems that include both tree convex and non tree convex instances.

In summary, for both random problems and structured problems, the acyclicity based algorithm has a clear performance advantage over the spanning tee based algorithm.

## 6 Conclusion

Polynomial algorithms have been designed to test tree convexity using ideas from consecutive ones property test and spanning tree. However, when the collection of sets is taken as a hypergraph, one can characterize the tree convexity by the acyclicity of the dual graph of the sets, which leads to a linear test algorithm thanks to the linear algorithm for testing the acyclicity of hypergraphs. In addition to its theoretical worst case efficiency, the acyclicity based algorithm is also very easy to implement and performs very well compared with the spanning tree based algorithm on the random problems we have generated. We notice that the algorithms to test row convexity (i.e., consecutive ones property) have been much more involved than the algorithm to test tree convexity although efforts have been made to find simpler algorithms (Habib et al., 2000; Meidanis et al., 1998). We are not aware of any work on consecutive ones property employing the properties of hypergraphs. It is interesting to investigate whether hypergraph properties and algorithms can help produce efficient and simple consecutive ones property test algorithms.

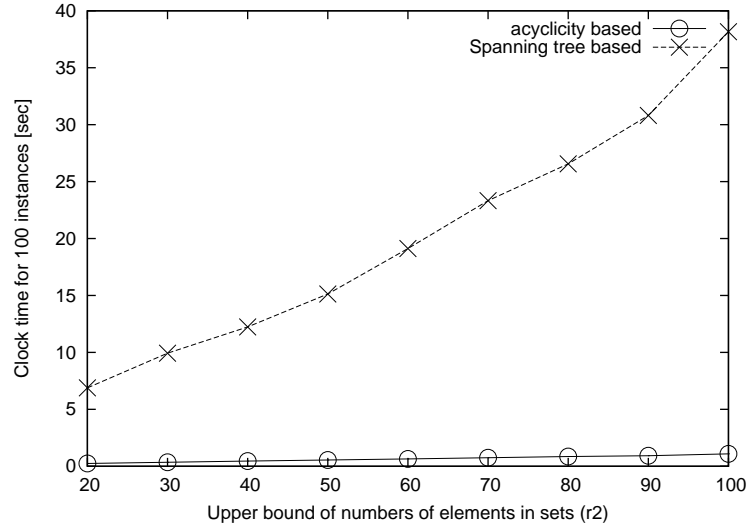


Figure 5: Performance of the algorithms on problems with  $\langle 50, 100, 2, r_2 \rangle$  with  $r_2$  varying from 20 to 90 with step 10.

Instance	Acyclicity based	Spanning tree based
arbitrary	0.58	34.67
arbitrary-npv	0.59	34.01
arbitrary-upv	0.59	34.91
matching	0.18	6.14
paths	0.29	16.27
regions	0.61	35.19
regions-npv	0.62	33.68
regions-upv	0.63	35.37
scheduling	0.17	42.38
L1	2.57	159.84
L2	4.04	324.02
L3	0.17	8.59
L4	0.16	6.95
L5	0.22	13.76
L6	0.29	18.43
L7	1.61	84.95
L8	0.62	8.8

Table 2: Performance of the algorithms on the benchmarking problems in (Leyton-Brown et al., 2000)

## Acknowledgment

We thank anonymous referees of earlier drafts of this work for pointing out to us Goodman and Shmueli's results (Goodman and Shmueli, 1983), and Gottlob and Greco's result (Gottlob and Greco, 2007).

## References

C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3): 479–513, 1983. ISSN 0004-5411. doi: <http://doi.acm.org/10.1145/2402.322389>.

- C. Berge. *Graphs and Hypergraphs*. American Elsevier Publishing Company, 1973.
- P. A. Bernstein and N. Goodman. Power of natural semijoins. *SIAM J. Comput.*, 10(4):751–771, 1981.
- K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976.
- V. Conitzer, J. Derryberry, and T. Sandholm. Combinatorial auctions with structured item graphs. In *AAAI*, pages 212–218, 2004.
- T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.
- P. Cramton, Y. Shoham, and R. Steinberg. *Combinatorial Auctions*. MIT Press, 2006.
- R. Dechter. *Constraint Processing*. Morgan Kaufmann, San Francisco, CA, 2003.
- R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.
- R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.
- E. Freuder. Synthesizing constraint expressions. *Communications of ACM*, 21(11):958–966, 1978.
- D. Fulkerson and O. Gross. Incidence matrices and interval graphs. *Pac. J. Math.*, 15:835–855, 1965.
- N. Goodman and O. Shmueli. Syntactic characterization of tree database schemas. *J. ACM*, 30(4):767–786, 1983.
- G. Gottlob and G. Greco. On the complexity of combinatorial auctions: structured item graphs and hypertree decomposition. In *ACM Conference on Electronic Commerce*, pages 152–161, 2007.
- G. Gottlob and S. Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems. *Comput. J.*, 51(3):303–325, 2008.
- M. Habib, R. M. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theor. Comput. Sci.*, 234(1-2):59–84, 2000.
- W.-L. Hsu. A simple test for the consecutive ones property. *J. Algorithms*, 43(1):1–16, 2002. ISSN 0196-6774. doi: <http://dx.doi.org/10.1006/jagm.2001.1205>.
- K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of 2nd ACM Conference on Electronic Commerce*, 2000.
- A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):118–126, 1977.
- J. Meidanis, O. Porto, and G. P. Telles. On the consecutive ones property. *Discrete Applied Mathematics*, 88(1-3):325–354, 1998.
- U. Montanari. Networks of constraints: fundamental properties and applications. *Information Science*, 7(2):95–132, 1974.
- D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976.
- M. H. Rothkopf, A. Pekec, and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.
- T. Sandholm and S. Suri. Bob: Improved winner determination in combinatorial auctions and generalizations. *Artif. Intell.*, 145(1-2):33–58, 2003.
- R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.
- P. van Beek and R. Dechter. On the minimality and global consistency of row-convex constraint networks. *Journal of The ACM*, 42(3):543–561, 1995.

- R. J. Wallace. Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPs. In *Proceedings of IJCAI-93*, pages 239–247, Chambéry, France, 1993. IJCAI Inc.
- G. Yosiphon. Efficient algorithm for identifying tree convex constraints. Manuscript, 2003.
- Y. Zhang and R. H. C. Yap. Consistency and set intersection. In *Proceedings of International Joint Conference on Artificial Intelligence 2003*, pages 263–268, Acapulco, Mexico, 2003. IJCAI Inc.